Python Cheat Sheet

by Roger Wattenhofer

Math

```
a = 10 // 3             # a = 3 (integer division)
b = 10 % 3              # b = 1 (remainder of division)
c = 10 / 3              # c = 3.3333333333333335 (automatic float)
d = (3 * 0.1) == 0.3    # d = False (floats are not exact!)
e = 2 ** 1000           # (power results in big number? no problem!)
1 + 1                   # → 2 (interactive: use _ for last result)
```

Strings & Lists

```
s = 'abcdefgh'
a = len(s)                      # a = 8
b = s[0]                        # b = 'a'
c = s[-1]                       # c = 'h'
d = s[1:3]                      # d = 'bc' (slicing)
e = s[3:-1]                     # e = 'defg'
f = s[3:-1:2]                   # f = 'df'
g = s[::-1]                     # g = 'hgfedcba'

x = list(range(5))              # x = [0, 1, 2, 3, 4] (range: lazy)
y = [3, 5, 8]                   # y = [3, 5, 8] (direct construction)
z = [i*i for i in range(1,6)]   # z = [1, 4, 9, 16, 25] (list compr.)
z.append(77)                    # (appending element to list)
z.extend([88,99])               # (extending list with another list)
```

Sets & Dictionaries

```
s = set({2,1,3,2,1})
s.add(4); s.remove(2)                               # s = {1, 3, 4}
print(3 in s)                                       # → True

d = {'alice': 24, 'bob': 22, 'charlie': 23}         # (dictionary)
d['eve'] = 26                                       # (add or change entry)
v = d.pop('charlie',None)                           # v = 23 (remove)
```

Control

```
if 2*x < y or x > 2*y:                     # if, elif, else as usual
  print("far")                             # no braces but indents
elif x == y:
  print("equal")
else:
  print("near")

for item in d:                             # traverse keys (also: list, set)
  for i in range(len(d)):                  # traverse over indexes
    for i, item in enumerate(d):           # both index and item
      break                                # break current loop

while x > 3:                               # while loops as usual
  print(x)
  x -= 1                                   # x = x - 1

print("hello") if x == 5 else print("x =",x)   # cond. expression
```

Functions

```
def square(x):                             # function definition
  return x*x                               # with return value

f = lambda x : x*x                         # lambda is one line function
print(f(5))                                # → 25
l = [i*i for i in range(1,6)]
l.sort(key=lambda v : v % 10)              # l = [1, 4, 25, 16, 9]
```

Scope

```
x = y = z = 1          # globally: set all = 1

def foo():
  global x             # keyword global: global x is used
  x, y = 2, 2          # x is global, y is local, z is unchanged
  print(x,y,z)         # → 2 2 1 (z is accessible)

foo()
print(x,y,z)           # → 2 1 1 (note that global y was unchanged)
```

Parameters

```python
def next_few(x, number = 3):           # default = 3
  res = []
  for y in range(number):
    res.append(x+y)
  return res

print(next_few(3))                     # → [3, 4, 5]
first, *middle, last = next_few(3,5)   # middle = [4, 5, 6], last = 7

def foo(var, *args, **d_args):
  print(var)                           # var is any type
  print("args =", args)                # *args is an arbitrary list
  print("d_args =", d_args)            # **d_args is a dictionary

foo(1, 2, 3, x=4, y=5)    # args = (2,3), d_args = {'x': 4, 'y': 5}
```

More Types

```python
b = True                    # boolean variables
bb = not b                  # bb = False
c = 2+3j                    # complex numbers
cc = c-2                    # cc = 3j
t = (2,3)                   # tuple, like a list but immutable
d = {t: True, c: False}     # keys cannot be lists (tuples okay)
s = str(c)                  # s = '(2+3j)' (conversion example)
```

Object Oriented

```python
class Foo:
  def __init__(self, name):  # constructor
    self.name = name

  def printName(self):       # method (self = always first argument)
    print('I am', self.name)

bar = Foo('bar')             # new object (constructor with name)
bar.printName()              # → I am bar
```

Copy

```python
old = [1,[2,3]]
same = old
shallow = old.copy()  # copying basic elements, referencing lists
import copy                    # deepcopy method must be imported
deep = copy.deepcopy(old)      # copying recursively
same[0] = 'a'
old[1][1] = 'c'
print('old =', old)            # → ['a', [2, 'c']]
print('same =', same)          # → ['a', [2, 'c']]
print('shallow =', shallow)    # → [1, [2, 'c']]
print('deep =', deep)          # → [1, [2, 3]]

def foo(v, l):                 # beware: changing lists in functions
  v += 1
  l[1] = 'x'

x = 5
ll = [1,2]
foo(x,ll)
print("function:", x, ll)      # → 5 [1, 'x']
```